البرمجة الإجرائية

Lecture No. 6

Number Representation

Matlab representation of floating point numbers

ميكاترونيك-سنة أولى-فصل أول

Dr. Eng. Essa Alghannam
Ph.D. Degree in Mechatronics Engineering

2024

مراجعة
تمثيل الأعداد الحقيقية في MATLAB

```
>> sin(pi)

ans =

   1.2246e-16

>> sind(180)

ans =

     0

>> sin(pi/2)

ans =

     1
```

- لا يمكن ترميز الأعداد بدقة مطلقة
- ترمز الأعداد في MATLAB بترميز الفاصلة العائمة ذو الدقة المضاعفة

مراجعة
تمثيل الأعداد الحقيقية في MATLAB

```
>> eps=1.e-6

eps =

   1.0000e-06

>> clear eps
>> eps

ans =

   2.2204e-16
```

sin(eps)

ans =

2.2204e-16

>> sin(0)

ans =

0

eps returns the distance from 1.0 to the next largest floating-point number.

مراجعة
تمثيل الأعداد الحقيقية في MATLAB

```
>> x=1; y=1+eps;z=1+eps/2;
>> isequal(x,y)

ans =

  logical

   0
```

Not equal

```
>> isequal(x,z)

ans =

  logical

   1
```

equal

```
>> isequal(y,z)

ans =

  logical

   0
```

Not equal

مراجعة
تمثيل الأعداد الحقيقية في MATLAB

```
>> eps(0)

ans =

 4.9407e-324

>> eps(1)

ans =

  2.2204e-16

>> eps(1000)

ans =

  1.1369e-13

>> eps(100000000)

ans =

  1.4901e-08
```

يعيد eps(x) أصغر عدد يضاف إلى x للحصول على عدد أكبر من x

مع إزدياد طويلة العدد، يزداد الفرق بينه و بين العدد التالي

```
>> 1 - 2/3 - 1/3

ans =

  5.5511e-17
```

```
>> 0.42-0.5+0.08

ans =

 -1.3878e-17

>> 0.08-0.5+0.42

ans =

   0
```

نتيجة العمليتين السابقتين تختلف بسبب عدم التمثيل الدقيق للأرقام و إختلاف النتائج البينية

مراجعة
تمثيل الأعداد الحقيقية في MATLAB

```
>> pi

ans =

   3.1416

>> pi/4000

ans =

   7.8540e-04
```

إذا كانت نتيجة عملية حسابية عدد صحيح، تظهر النتيجة على شكل عدد صحيح

إذا كانت النتيجة عدد حقيقي، تظهر النتيجة بأحد الشكلين التاليين:

- يظهر 4 أرقام بعد الفاصلة العشرية
- تظهر النتيجة بالتمثيل العلمي

خلاصة Floating-Point Numbers

- MATLAB® represents floating-point numbers in either double-precision or single-precision format. The default is double precision, but you can make any number single precision with a simple conversion function.

Double-Precision Floating Point:

Double precision, requires 64 bits

2^1024=1.7976e+308

realmax

```
>> realmax('double')

ans =

  1.7977e+308
```

```
>> realmax

ans =

  1.7977e+308
```

Bits	Usage
63	Sign (0 = positive, 1 = negative)
62 to 52	Exponent, biased by 1023
51 to 0	Fraction f of the number 1.f

Single-Precision Floating Point:

SinglePrecision, requires 32 bits

2^128=3.4028e+38

$1.11111111111111111111111_2 x 2^{127} = 3.4028235_{10} x 10^{38}$

```
>> realmax('single')

ans =

  single

  3.4028e+38
```

Bits	Usage
31	Sign (0 = positive, 1 = negative)
30 to 23	Exponent, biased by 127
22 to 0	Fraction f of the number 1.f

جَامعة المَنارة
MANARA UNIVERSITY

خلاصة

Floating-Point Numbers

Because MATLAB stores numbers of type single using 32 bits, they require less memory than numbers of type double, which use 64 bits. However, because they are stored with fewer bits, numbers of type single are represented to less precision than numbers of type double.

- Use double for numbers greater than approximately 3.4×10^{38} or less than approximately -3.4×10^{38}.
- For numbers that lie between these two limits, you can use either double- or single-precision, single requires less memory.

خلاصة Floating-Point Numbers

Creating Double-Precision Data

Because the default numeric type for MATLAB is double, you can create a double with a simple assignment statement:

```
>> x = 25.783;
```

The whos function shows that MATLAB has created a 1-by-1 array of type double for the value you just stored in x:

```
>> whos x
  Name      Size                Bytes  Class

  x         1x1                     8  double
```

To verify that x is a floating-point number.

```
>> isfloat(x)
ans =

  logical

   1
```

Create a 64-bit integer

```
>> y = int64(-589324077574); % Create a 64-bit integer
>> isfloat(y)

ans =

  logical

   0
```

```
>> x = double(y) % Convert to double

x =

   -5.8932e+11

>> isfloat(x)

ans =

  logical

   1
```

خلاصة Floating-Point Numbers

- **Creating Single-Precision Data**

```
>> x = single(25.783);
>> whos
  Name      Size            Bytes  Class     Attributes

  x         1x1                 4  single
```

```
>> y = int64(-589324077574);
 % Create a 64-bit integer

x = single(y)
 % Convert to single

x =

  single

  -5.8932e+11
```

```
>> isfloat(y)

ans =

  logical

   0
```

```
>> isfloat(x)

ans =

  logical

   1
```

```
>> class(y)

ans =

  'int64'
```

```
>> class(x)

ans =

  'single'
```

Rules for arithmetic operations using different classes

- **Rule 1:** arithmetic operations are allowed between two variables that have the same class. The variables could be a scalar, a vector or an array. For example,

```
a = int8(-3);
b = int8(20);
c = a + b;
```

- Another example,

```
r = int16(1:5);
s = int16(6:10);
t = r + s;
```

- **Rule 2:** arithmetic operations are not allowed between an integer class (signed or unsigned) and other classes, except for the double class. This rule is valid only for scalar variables.

  Examples of allowed operations

  ```
  x = int8(-3);
  y = 20;
  z = x + y;
  ```

- Another example,

  ```
  x = uint8(-3);
  y = double(20); %Notice that the command y = double(20) is equivalent to the command y = 20.
  z = x + y;
  ```

- Note that in these examples **the variable z** that stores the result still keeps the **original integer class.**

```
>>class(z)
ans =
uint8
```

Examples of operations that are not allowed are;

```
>>x = int8(-3); y = uint8(20); z = x + y;
>>x = uint8(3); y = uint16(20); z = x + y;
>>x = int8(-3); y = int32(20); z = x + y;
>>x = int8(-3); y = single(20); z = x + y;
>>x = int8(1:4); y = double(1:4); z = x + y;
```

- Note that in this last example, although operations between integer and double classes **are** allowed, this is only true for scalars, and here we have vectors! – therefore this last example is also **not** allowed

- **Rule 3:** arithmetic operations are allowed between the single class and the double class. This rule is valid for scalar, vector and array variables.

- An example of allowed operations:

```
x = single(-3);
y = 20;
z = x + y;
```

- Another example,

```
r = single(1:4);
s = double(1:4);
t = r + s;
```

The variable `t` that stores the result has a `single` class.

```
>>class(t)

ans =
single
```

- **Rule 4:** Once a vector or an array variable of a specific class is created, assigning values of a different class to elements of that variable does not change its class. This rule is illustrated by the following example.

```
>>x = uint8(1:10)
x =
1 2 3 4 5 6 7 8 9 10

>>class(x)
ans =
uint8
```

```
>>x(3) = int8(11)
x =
1 2 11 4 5 6 7 8 9 10

>>class(x)
ans =
uint8
```

جَامعة
المَنارة
MANARA UNIVERSITY

Thanks .